Transbase Relational Database System

Version 5.3.1

UFI User's Guide

Copyright © 1987 - 2003 by:

Transaction Software GmbH
Thomas-Dehler-Str 18
D-81737 Munich
Germany
Telephone: 0 89 / 6 27 09 0
Telefax: 0 89 / 6 27 09 11
Electronic Mail: tas@transaction.de

1 TransBase User Friendly Interface 5
1.1 Overview 6
1.2 Syntax Notation 7

2 Starting UFI 8
2.1 Startup Feature 8
2.2 Shell variables 10
2.3 Terminal capabilities 11

3 The menu system 12
3.1 Main Menu 13

4 UFI editor 15
4.1 Editing commands 17
4.2 Entering special characters 19
4.3 Restrictions 20

5 Procedure management 20
5.1 Procedure Submenu 22

6 Setting options 24
6.1 Set command 24
6.1.1 Set 24
6.1.2 Autocommit 24
6.1.3 Backup files 26
6.1.4 Timer 26
6.1.5 Consistency 27
6.1.6 Column width 27
6.1.7 Representation of null values 27
6.1.8 Editor 28
6.2 Options menu 29

7 Displaying query results 29
7.1 Moving the display window 32
7.1.1 Move the display window forward 32
7.1.2 Move the display window backward 32
7.1.3 Move the display window to the right 32
7.1.4 Move the display window to the left 32
7.1.5 Direct positioning 33
7.1.6 Display of extra long attributes 33
7.2 Format commands 34
7.2.1 Left aligned output of attribute values 34
7.2.2 Right aligned output of attribute values 34

7.2.3 Change column width 34
7.2.4 Change decimal places 35
7.2.5 Show leading zeros 35
7.2.6 Change column headers 35
7.2.7 Change the display of NULL values 35
7.2.8 Change column separator 36
7.2.9 Exclude columns 36
7.2.10 Include columns (reverse of exclude command) 37
7.3 Output commands 38
7.3.1 Store query result into a file 38
7.3.2 Display Command 38
7.4 Quit command 38

8 Procedures and Transactions 38
8.1 Text variables 42
8.2 Calling other procedures 43
8.3 Transaction control 44
8.4 Database Control 46
8.5 Changing Current Directories 47
8.6 Setting Timeout 47

9 Error handling and trouble shooting 48

1 TransBase User Friendly Interface

UFI is an acronym for 'User Friendly Interface', since it provides an interactive and menu oriented interface between the user at the terminal and the database system TransBase. UFI extends pure TransBase Query Language (TB/SQL) and makes it much more powerful and easier to use. The main features are:

A sophisticated procedure concept, which allows to compose complex transactions without the need to write an application program in 'C'. Procedures consist of TB/SQL statements, of both data definition and data manipulation language, mixed with user interactions.

It offers a powerful and very useful parameter mechanism, which is solely missing in pure TB/SQL.

A built-in screen editor simplifies the task of creating and editing procedures.

Procedures are passed to TransBase for evaluation. In case of a "select" statement a set of tuples is returned to UFI, which is displayed on the screen. The user can interactively choose the output format and print (or store) the result.

Procedures can be saved in ordinary text files, gathered in a separate directory, for repeated execution at a later time.

UFI has a number of system directives which affect its operating mode. Default values for these options can be set through a submenu or at startup time from a file.

UFI is implemented as a normal application program, i.e. there are actually two processes running (UFI and TransBase) exchanging requests and results.

1.1 Overview

This manual describes UFI for UNIX System V and Berkeley UNIX 4.2 BSD (respectively ULTRIX as called by DEC). Where UNIX V and Berkeley UNIX differ, we will give the appropriate commands for both UNIX versions. We also assume that users work with either 'sh' or 'csh'.

Although we are making thorough use of examples, this manual is not intended as a tutorial for relational database systems in general or TransBase in particular. The reader should have at least a basic knowledge of relational databases, the TransBase query (DML) and data definition (DDL) language TB/SQL (see TransBase TB/SQL Reference Manual) and of course the UNIX operating system.

Section 2 of this manual describes, how to start UFI in order to access databases. Section 3 gives an overview of the menu organization. Sections 4 to 7 describe the built-in editor and each submenu in detail. Section 7 also contains the commands of the display formatting facility. In section 8 we handle the execution of procedures (queries). This covers how to start and commit transactions, how to cancel troublesome queries and how to parameterize your procedures and call other procedures. Section 9 describes the output of error messages, detected by UFI and TransBase, also giving some hints to avoid the most common mistakes.

1.2 Syntax Notation

We use a concise notation to describe the syntax of all commands:

Most keywords can be abbreviated to any unique prefix; e.g. the notation

```
col[umn]
```

means that 'col' is a unique prefix and 'umn' is optional and need not be typed.

```
col, colu, colum
```

are all legal abbreviations of the keyword above. Keywords are not case sensitive, so you may also write

```
COL
Column
```

All variable parts of a statement are denoted in brackets '< ... >'.

<n> means a positive number. Examples are:

```
1
20
5
```

<text> means a string of characters. The string must be put into double quotes, when <text> conflicts with a keyword or contains special symbols, e.g. ';' or blank. Examples are:

```
Address
"name"
"  |  "
```

<filename> or <database> denotes the name of a file or database directory, respectively. For <filename> the same quoting rules apply as for <text>. Examples are:

```
myfile
../../nextdir/file1
"format"
```

Many commands are applied to particular columns of a query result shown on the screen. <co> denotes such an identifier of a column. There are two ways to identify columns:

First, columns can be selected by their position, that's where they appear in the "select" clause of a query. Then <co> is a positive number as described in (3). Even when columns are invisible, because they are excluded from display, their position remains unchanged.

Second, a column can be identified by its header or any prefix of the header. An ambiguous prefix selects the first column, whose header matches the prefix. Then <co> is a text as described in (4) with the same quoting rules. Examples are:

```
6
1
2
suppno
"Name"
```

<colist> is a list of column identifiers separated by blanks and enclosed in parentheses. Column headers and numbers may be mixed in the list. Instead of a <colist> you may also specify a single column as described in (6). Examples are:

```
(4 cname 3)
("col" cname cpos)
1
cname
```

Control characters are special keys of your keyboard. We are using three different notations for control characters and function keys:

^X	press CTRL key together with letter key X
RETURN, LF, ESC	press corresponding key of your keyboard
:xx:	(denotes function key named xx)

2 Starting UFI

Before starting UFI you must have installed a database, either by creating a new database (see the description of tbadmin in TransBase System Guide) or by accessing an already existing database.

To start UFI enter the command:

ufi [*dbname*] ...

where *dbname* denotes the logical name of a local or remote database.

2.1 Startup Feature

When FI is invoked it searches for a file ".ufirc" located in the current working directory, which will be executed every time you call UFI. The .ufirc file is handled like an ordinary procedure (see section 8).

This serves to set some of UFI's directives or to execute particular queries at startup time. The following figure shows a typical example, where all options are set to their default values from the .ufirc file.

Sample File .ufirc

```
set autocommit proc;
set backup last;
set timer off;
set consistency high;
set string_width 40;
set int_width 30;
set real_width 10;
set str_nulls "?";
set num_nulls "?";
set dateformat USA;
set editor ufi;
```

The startup feature is completely optional. No error message is given if the .ufirc file is missing.

2.2 Shell variables

UFI and TransBase access the following shell variables:

transbase:

directory where the database system TransBase resides. This has been established by your system administrator at the installation phase. In the example given below we assume that the database system resides in the directory /usr/TransBase.

term:

The type of your terminal (e.g. TERM=VT100)

path:

A list of directories where the shell searches for executable programs. The PATH variable should read somewhat like

```
PATH="/bin:/usr/bin: ... :$TRANSBASE"
```

Note:
The list of pathnames is enclosed in double and not in single quotes.

editor:

The name of your favorite text editor (e.g. EDITOR=vi)

shell:

The name of your favorite shell. In most cases this is either the Bourne Shell (SHELL=/bin/sh) or the C-Shell (SHELL=/bin/csh).

To setup the environment variables properly enter the following lines into your .profile (sh) or .login (csh) file, respectively:

FILE .profile:

```
TERM=<terminal type>; export TERM
TRANSBASE=/usr/transbase; export TRANSBASE
PATH="<anything you like>:$TRANSBASE"; export PATH
EDITOR=emacs; export EDITOR
SHELL=sh; export SHELL
```

FILE .login:

```
setenv TERM <terminal type>
```

```
setenv TRANSBASE /usr/transbase
setenv PATH "<anything you like>:$TRANSBASE"
setenv EDITOR vi
setenv SHELL csh
```

Normally, the TERM variable is set automatically during login and you may omit the first line of each file.

2.3 Terminal capabilities

UFI uses the terminal type as set in the TERM variable to select the specific features of your terminal from the terminfo (UNIX V) or termcap (Berkeley UNIX) library, respectively. UFI works fine with terminal sizes of 24 (or 25) lines x 80 characters. It also takes advantage of more lines and broader screens, but the menus will look somewhat odd. The terminal must have the following capabilities (otherwise you may get garbage on your screen):

Name	Description of Capability
al	Add new blank line
cd	Clear to end of display
ce	Clear to end of line
cl	Clear screen
cm	Cursor motion
dc	Delete character
dl	Delete line
ic	Insert character
ks	Start "key pad transmit" mode
ke	End of "key pad transmit" mode
nd	Non-destructive space (cursor right)
so	Start stand out mode
se	End stand out mode

Additionally, UFI looks if there are function keys defined for the terminal.

Name	Description of function key
kr	Sent by "right arrow" key

kl	Sent by "left arrow" key
ku	Sent by "up arrow" key
kd	Sent by "down arrow" key
kb	Sent by "backspace" key
kh	Sent by "home" key
cd	Sent by "clear" key
ic	Sent by "insert" key
dc	Sent by "delete" key
al	Sent by "insert line" key
ce	Sent by "erase line" key
dl	Sent by "delete line" key

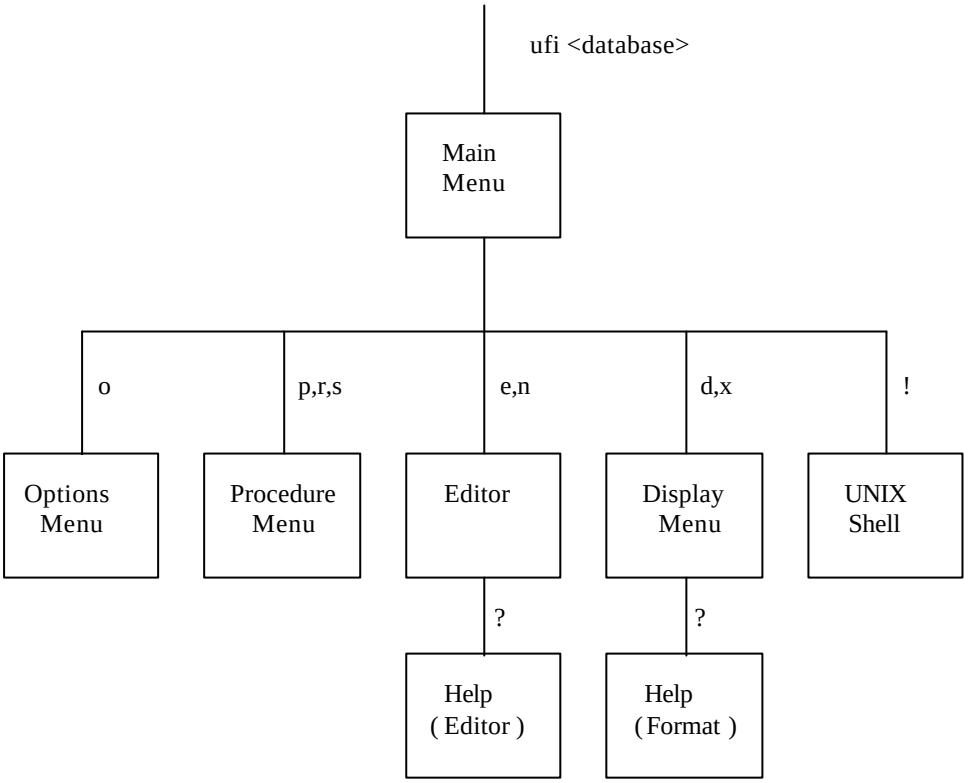
The function keys are not mandatory, but when defined you can use the arrow and other function keys of your keyboard instead of the control characters (see section 4).

Note:

When function keys are defined for your terminal in the terminfo (termcap) library, sometimes strange effects occur, since a function key might redefine some of the control characters normally used by UFI.

3 The menu system

The next figure shows the overall organization of the menus:



The menu system consists of:

- the main menu,
- the option submenu,
- the procedure management submenu,
- the editor
- and the submenu to display query results.

Executing a "select" query automatically branches to the display submenu.

Any submenu is invoked by pressing the associated (lower case) letter. Just hit the ESC key to leave a submenu or the editor and return to the main menu. Don't press the RETURN key except when you are asked for confirmation.

The following sections describe each menu in detail.

3.1 Main Menu

The main menu is shown in the next figure.

DB: sample	**COMMIT_P**	**File:**
edit procedure	e	procedure_management p,r,s
edit new procedure	n	
execute procedure	x	display last result d
Connect database	C	Disconnect database D
exit UFI	q	options o
abort transaction	a	commit transaction c
No Transaction active		Host: host

The top line is a status line showing the name of the database, the name of the procedure file you are currently working with, and the commit mode. The bottom line shows the current state of your transaction, error messages and warnings.

The 'a' and 'c' selections are only visible if applicable.

The menu offers the following selections:

n These command are used to invoke the editor. With the "e" command the last procedure can be edited again, while the "n" command clears the editor buffer and you can enter a new procedure. The previous contents of the editor buffer is saved in a backup file. The backup files are named <database>/query/editbuf<n>.bak, where <n> is a running number. This means that you can restore (via the procedure management submenu) any of the procedures you have created during a UFI session.

Note: When you start UFI for the next time any previous backup files will be deleted automatically.

x The "x" key starts the execution of a procedure. The contents of the editor buffer is passed query by query to the database system. Section 8 will tell you more about the execution of procedures and transactions.

o The "o" key invokes the options submenu. The options menu is used to set default values for UFI directives.

s These keys invoke the procedure submenu. This menu shows a list of all stored procedures. Using the "p" key invokes the procedure submenu where you stay until explicitly leaving the menu by ESC key. The procedure menu offers the following choices:

- to read (recall) a procedure into the editor buffer,

- to store the contents of the editor buffer into a procedure,
- to delete a procedure,
- to display (page) a procedure at the terminal.

The main menu keys "r" and "s" can be considered as an abbreviation for recalling or storing single procedures.

ESC The "q" or ESC key is used to quit UFI and return to the shell. The corresponding item will be highlighted and you are asked for confirmation. Any other key but RETURN will undo the quit command. You are also warned about an open transaction, since it would have to be aborted when you leave UFI.

d The "d" key invokes the display menu and shows you the result of the last query once again.

a If the autocommit mode is switched off, the main menu contains two additional items, namely "c" and "a". The "c" command is used to commit a transaction while the "a" command will abort it. When you press the "c" or "a" key, the corresponding item will be highlighted and you are asked for confirmation. Any other key but RETURN will undo the command, i.e. not commit or abort the current transaction.

D The "C" key is used to connect and/or login to a database. The user is prompted for the logical name of the database, his username and his password. If a user has more than one active connection, the switch statement is used to switch from one database to another. The "D" key disconnects the user from the currently active database.

! Pressing "!" suspends UFI and puts you into an interactive UNIX shell, thus you can intermediately execute any UNIX commands you like. To resume your UFI session you have to exit the shell, e.g. by typing '^D'. The name of the shell is determined by the environment variable **SHELL**.

4 UFI editor

The built-in editor is a screen oriented editor for creating and correcting procedures. The editor doesn't directly operate on procedure files but uses a small memory resident buffer, called the editor buffer, to hold the text of the edited procedure. The size of the editor buffer is restricted to 300 lines.

Upon entering the editor the screen contains a top status line shown below:

```
UFI  LINE:  1  COL: 1   INSERT   Leave with ESC, Help = ?
```

You enter text just by typing it. If the insert mode is switched on, the characters typed at the keyboard will be inserted at the cursor position. If the insert mode is off, the entered text will overwrite the characters under the cursor.

4.1 Editing commands

The following table summarizes the commands supported by the editor. Alternative keys to invoke an editor function are separated by blank. Function keys (see section 2.3) as defined in the terminfo (respectively termcap) are denoted as :xx:.

^S :kl: :kb:	moves the cursor one character to the left
^D :kr:	moves the cursor one character to the right
^A	moves the cursor one word to the left
^F	moves the cursor one word to the right
^B	moves the cursor alternatingly to the first and last column of a line
^E :ku:	moves the cursor one line up
^X :kd:	moves the cursor one line down
^R	moves the cursor one page up
^C	moves the cursor one page down
^W	scrolls screen down one line
^Z	scrolls screen up one line
^O :ic:	toggles insert mode on/off
<DEL>	deletes character to the left
^G :dc:	deletes character under the cursor
^T	deletes the word to the right
^K :ce:	deletes all text from the cursor to the end of the line
^Y :dl:	deletes the complete line
^N :al:	inserts a new blank line at the cursor position
^P	enter next character literally (see section 4.2)
?	shows a list of these editor commands
ESC	quits the editor and return to the main menu.

Note:

Editing is done entirely in memory and any associated file is not affected. In order to make changes permanent the edit buffer must be saved explicitly into a procedure (see the **s** item from the main menu).

4.2 Entering special characters

There are some characters that are interpreted in a special way by either the editor, UFI or TransBase:

'?' If you want to enter a '?', you must type '^P' followed by '?'.

'^x' All control characters are special symbols for the editor. To enter a control character as ordinary text you must first press '^P' and then the control character.

' ' The single quotes are used to delimit string constants in TB/SQL statements. If you need a string with quotes inside, you must double the quote. E.g.

```
'This is a string with quotes " inside'
```

';' The semicolon is interpreted by UFI as a delimiter between statements of a procedure. If you need a ';' which should not be interpreted in this way, especially in string constants, you must escape it with '\'. E.g.

```
'TransAction Software\; Munich'
```

'$' The dollar sign is interpreted by the UFI as string expansion character (see section 4). If you need the dollar sign literally, escape it with '\'. E.g.

```
\$var\$
```

'\' To enter the escape character itself, you must escape it with '\'. E.g.

```
'This is a string with a backslash \\ inside'
```

4.3 Restrictions

The UFI editor has the following restrictions:

- There is no command mode e.g. to search and substitute strings. If you need such features you may replace the built-in editor by your favorite system editor by the UFI directive:

  set editor system

- The length of a line is limited to 80 characters. When inserting text into a line, the rest of the line is not moved into the next line automatically. Instead, an error "LINE TOO LONG" is signaled in the status line.

- The editor buffer cannot hold more than 300 lines.

These restrictions also apply, when you read in (recall) files created with other editors. Lines longer than 80 characters will be split into several lines and only the first 300 lines of a file will be read.

5 Procedure management

UFI supports user-defined procedures containing any sequence of TB/SQL, DDL or UFI statements. Procedures can be created, edited (by the editor described in chapter 4), stored, recalled and deleted.

Those procedures are searched in the current "query directory". If not explicitly changed, the current query directory is the current working directory of the UFI session. I.e. if you call UFI from your home directory the current working directory and the current query directory will be your home directory. The "cdq" statement can be used to change the current query directory without affecting the current working directory.

The recall command fetches a permanently stored procedure into the editor buffer, thus allowing the user to edit or execute this procedure.

Conversely, the store command saves the contents of the editor buffer into a permanent procedure, which can be recalled in later sessions.

All procedures are stored as files with the extension .sql; conversely, only files with the extension .sql are shown in the procedure menu.

Note:

The editor buffer is not saved automatically into a permanent procedure, as mentioned before. But it is saved into a temporary backup file named

editbuf<n>.bak

where <n> stands for a running number.

This backup file will not be removed. Instead, it is reused if UFI is called from the same query directory.

5.1 Procedure Submenu

The figure below shows an example of the procedure management submenu.

delete = d store = s recall = r page = p		
1 .ufirc	2 compl	3 complquery
4 corr	5 dbprak	6 f
7 h	8 join1	9 join2
10 join3	11 join4	12 join5
13 join6	14 loc0	15 loc1
16 loc2	17 loc3	18 make.db
19 make.statistic	20 makex.db	21 test
22 proj	23 query	24 query1
25 relation	26 split1	27 split2
>> Scroll down with ^C, up with ^R Leave with ESC		

What you see is a list of all procedures stored in the current query directory. The directory may contain more files than can be displayed on one screen. As within the editor use the keys '^C' and '^R' to scroll forward and backward through the directory.

The top line of the menu lists possible operations:

d delete a procedure

s store the editor buffer into a permanent procedure

r read (recall) a procedure into the editor buffer. If you have edited a procedure before without having saved it, this procedure will be saved into a temporary procedure as described in section 5.

p display (page) a procedure at the terminal

After selecting an operation you will be prompted for the procedure you wish to operate on. A procedure can be selected in two ways: by entering its full name or by entering a number associated with each procedure.

Note:

Association between procedure names and numbers may change when procedures are deleted or edited.

The shortcut **r** within the main menu stands for the selection sequence **p** (invoke the procedure submenu) followed by **r** (recall a procedure) and followed by ESC (return into the main menu). Similarly, **s** is a shortcut for the sequence **p**, **s** and ESC.

6 Setting options

UFI has a number of directives which affect its operating mode. The options may be altered by the user to accommodate operation of UFI to his own requirements. Each option can be changed either within procedures by the **set** command or interactively via the options menu. Section 6.1. describes the set command and in section 6.2. the options menu is explained.

6.1 Set command

The set command must be placed in a procedure. Normally, you put a series of set commands into the .ufirc start up file, though initialization of UFI's system parameters takes place at the beginning of a session.

6.1.1 Set

A set command without any argument (or invalid argument) gives a list of valid options and their current settings. E.g.

Example:

```
set

displays the list

          autocommit = proc
          backup = last
          timer = off
          consistency = high
          string_width = 20
          integer_width = 10
          real_width = 15
          str_nulls = <empty string>
          num_nulls = <empty string>
          dateformat = USA
          editor = ufi
```

Note:

The last example summarizes the default settings of UFI at the beginning of a session assuming the user hasn't changed them.

We will now discuss each directive in more detail.

6.1.2 Autocommit

Syntax:

```
set autocommit proc
set autocommit query
set autocommit expl
```

When autocommit is switched to proc, each procedure, when executed, is treated as a transaction which is automatically committed at successful completion and aborted otherwise.

When autocommit is set to expl, the user himself must commit a transaction. This offers more freedom on transaction handling. Manual transaction control on the other side may produce long transactions, which would hold locks and thus may reduce the concurrency of other transactions.

When set to query, each statement of a procedure is a transaction of its own and is autocommitted if successful. Note that a sequence of update statements is no longer atomic.

Default:

```
autocommit proc.
```

Note:

When autocommit is switched to expl, two additional items, i.e. 'commit' and 'abort', will appear in the main menu.

Warning:

When autocommit is switched from expl to proc or query and a transaction is active at the moment, this transaction will be committed automatically.

6.1.3 Backup files

Syntax:

```
set backup all
set backup last
```

When backup is toggled to 'all', backup files of procedures are all saved during a session and consecutively numbered as editbuf<n>.bak.

When backup is toggled to 'last', only the backup file of the last procedure is saved. This saves some time and disk space when you write many ad hoc queries which are never needed again.

Default:

```
only the last procedure is saved.
```

6.1.4 Timer

Syntax:

```
set timer on
set timer off
```

The timer is provided for performance tests to measure the time it takes to evaluate a specific query.

Note:

When time measurement is switched on, you cannot enter any command in the display menu before the complete query result has been computed.

Default:

```
Time measurement is off.
```

6.1.5 Consistency

SYNTAX:

```
set consistency low
set consistency high
```

NOTE:

The setting of consistency influences the unlock strategy of TransBase.

EFFECT:

When consistency is toggled to high, read locks will be held until the end of the transaction. When toggled to low, read locks will be held until the end of the query. Update locks and exclusive locks will always be held until the end of the transaction.

DEFAULT:

```
high consistency
```

6.1.6 Column width

Syntax:

```
set integer_width <n>
set real_width <n>
set string_width <n>
```

NOTE:

The option column width specifies the number of characters UFI will use to display attributes of type int, real/numeric and string, respectively. You may want to increase or decrease <n> if the attribute values in the database tend to be longer or shorter than the default settings.

In the display menu you can override these settings for single columns.

Defaults:

```
character for type integer
15 character for type real/numeric
20 character for type string
```

6.1.7 Representation of null values

Syntax:

```
set num_nulls <text>
set str_nulls <text>
```

This entry specifies the pattern which will be displayed instead of a null value. This pattern is different for attribute class integer/real/numeric on the one hand and class string on the other hand.

Default:

```
the empty string for both kinds of null values.
```

6.1.8 Editor

Syntax:

```
set editor ufi
set editor system
```

With this option you select the editor which is invoked from the main menu. Either the UFI built-in editor (see section 4) or your favorite UNIX editor may be used. The name of the system editor is read from the shell variable **EDITOR**. When such a shell variable is not defined, the editor defaults to vi.

The system editor is called with the name of a temporary file as its first argument. The temporary file contains the current procedure.

Note:

> While you can edit larger files with the system editor, still the same restrictions apply as for the UFI editor when this file is read in for execution.

6.2 Options menu

Below the options menu is shown.

UP = ^E DOWN = ^X LEFT = ^S RIGHT = D		
Automatic	PROC	String_width 20
Timer	OFF	Real_width 15
Backup	LAST	Integer_width 10
Dateformat	ISO	
Consistency	HIGH	Str_Char_Nulls
Editor	ufi	Int_Real_Nulls

An option is selected by moving the cursor onto the corresponding item using the cursor control characters or function keys as described for the editor. When you have put the cursor on an item, you see a message in the last line, which asks you either to enter a number or text, or to press RETURN to toggle an option.

The autocommit toggle may be set to one of PROC, QUERY, EXPL. PROC means that transactions are committed after each procedure. QUERY means that transactions are committed after each query. EXPL means that transactions are not committed automatically by UFI, i.e. it is left up to the user when to commit or abort a transaction.

The Dateformat toggle takes one of the following settings: ISO, USA, EUR, OWN. If set to OWN, the user has the possibility to define his own format string. See the routine tb_dt_format in the TB/X manual for a description of the format string.

7 Displaying query results

As soon as UFI executes a "select" query the display menu is activated automatically.

Example:

The query

```
select * from systable s, syscolumn t
where s.tname=t.tname
```

will produce a screen as follows.

tname	ttype	segno	colno

Statistics	R	13	8
Statistics	R	13	8
Statistics	R	13	8
Statistics	R	13	8
Statistics	R	13	8
Statistics	R	13	8
Statistics	R	13	8
inventory	R	7	3
inventory	R	7	3
inventory	R	7	3
quotations	R	9	5
quotations	R	9	5
quotations	R	9	5
quotations	R	9	5
quotations	R	9	5
==>			
Leave with ESC, Help = ?			
Max. Tuple: 19 First Tuple: 1			

Note:

The display of tuples will be somewhat deferred, because UFI must wait until TransBase has computed the next tuple of the result.

At the top of the screen the names of the attributes are displayed. The last attribute is suffixed with '|'. At most 15 tuples are displayed on the screen. The last tuple of the result is succeeded by a line of '=='.

The bottom of the screen consists of a command input line and a status line. The status line shows, how many tuples the result has, and on which tuple the display window is positioned at the moment. In our example the result has 19 tuples and we are standing at the first tuple. On the command line you can enter commands to move the display window through the set of tuples and to format the screen output.

The command line supports some basic editing features. These are a subset of the editor functions:

^Z	restores the previous command line
^S :kl: :kb:	moves the cursor one character to the left
^D :kr:	moves the cursor one character to the right
^B	moves the cursor alternatingly to the first and last character of a line
DEL	deletes character to the left
^G :dc:	deletes character under the cursor
^K :cd:	deletes all text from the cursor to the end of the line
^Y	deletes the complete line
?	shows syntax of available display commands

The following paragraphs describe the display commands in full detail.

7.1 Moving the display window

In many cases the result will not fit on the screen. The screen will not be wide enough to show all attributes or it will not be long enough to show all tuples. There are commands for each direction to move the display window through the query result.

7.1.1 Move the display window forward

Syntax:

```
forw[ard]
forw[ard] <n>
forw[ard] max
```

The first form moves the display window down one screen (about 15 tuples). The cursor down key (i.e '^X' and :kd:) has the same effect. The second form moves the display window forward <n> tuples. The third form moves the display window to the last tuple.

7.1.2 Move the display window backward

Syntax:

```
ba[ckward]
ba[ckward] <n>
ba[ckward] max
```

The first form moves the display window up one screen (about 15 tuples). The cursor up key (i.e '^E' and :ku:) has the same effect. The second form moves the display window backward <n> tuples. The third form moves the display window to the first tuple.

7.1.3 Move the display window to the right

Syntax:

```
r[ight]
r[ight] <n>
r[ight] max
```

The first form moves the display window one screen to the right, the last column on the previous screen becoming the first column. The second form moves the display window <n> columns to the right. The third form moves the display window to the last column.

7.1.4 Move the display window to the left

Syntax:

```
l[eft]
```

```
l[eft] <n>
l[eft] max
```

The first form moves the display window one screen to the left, the first column on the previous screen becoming the last column. The second form moves the display window <n> columns to the left. The third form moves the display window to the first column.

7.1.5 Direct positioning

Syntax:

```
col[umn] <n>
```

The display window is moved to the <n>-th column, which becomes the first column on the screen.

7.1.6 Display of extra long attributes

Syntax:

```
ta[b] <n>
ta[b]
ta[b] all
```

The TAB command operates only on the first column of the screen which must be a string attribute. The first form allows you to shift the string so that the first <n> characters are skipped when displaying it. The second form displays the string from the beginning. The third form is another way to display extra long strings. The first column is extended to the full width of the screen and the string wraps over several lines. You will prefer the 'tab all' command to see the complete string.

7.2 Format commands

You may change the output format of the result to adapt it to your personal taste. This section describes the format commands you can apply.

7.2.1 Left aligned output of attribute values

Syntax:

```
[format] col[umn] <colist> ml[eft]
```

All columns specified in <colist> are displayed left aligned.

Default:

Attributes of type string are displayed left aligned

Attributes of type number are displayed right aligned

7.2.2 Right aligned output of attribute values

Syntax:

```
[format] col[umn] <colist> mr[ight]
```

All columns specified in <colist> are displayed right aligned.

Defaults:

Attributes of type string are displayed left aligned

Attributes of type number are displayed right aligned

7.2.3 Change column width

Syntax:

```
[format] col[umn] <colist> w[idth] <n>
```

All columns specified in <colist> are displayed with a width of <n> characters (<n> between 1 and 78).

Default:

the values from the options menu.

7.2.4 Change decimal places

Syntax:

```
[format] col[umn] <colist> d[places] <n>
```

All columns specified in <colist> must be of type real or numeric. They are displayed with <n> decimal places (<n> between 1 and 16).

Default:

```
= 10.
```

7.2.5 Show leading zeros

Syntax:

```
[format] col[umn] <colist> z[eros] on
[format] col[umn] <colist> z[eros] off
```

All columns specified in <colist> must be of class number. They are displayed with leading zeros (on) or without (off).

Default:

without leading zeros (off).

7.2.6 Change column headers

Syntax:

```
[format] col[umn] <co> na[me] <text>
```

The header of the specified column <co> is changed to <text>.

Default:

the attribute name coming from TransBase.

7.2.7 Change the display of NULL values

Syntax:

```
[format] nul[ls] <text>
[format] nul[ls] str[ing] <text>
[format] nul[ls] num[ber] <text>
```

Attributes having the NULL value are displayed as <text>. You may define different representations for NULL values of attributes of either attribute class string or of class number. This overrides only the representation of null values for this query.

Default:

the values from the options menu.

7.2.8 Change column separator

Syntax:

```
[format] sep[arator] <n> bl[anks]
[format] sep[arator] <text>
```

This format command specifies which string will be printed to separate columns on the screen. It also serves to vary the distance between columns, because <text> may contain up to 30 characters.

Default:

one blank.

7.2.9 Exclude columns

Syntax:

```
[format] ex[clude] <colist>
[format] ex[clude] all but <colist>
```

With the first form all columns specified in <colist> are excluded from display. This means they are no longer shown on the screen or will be printed, when you issue an output command. With the second form all columns but those specified in <colist> are excluded from display.

Default:

no column is excluded.

7.2.10 Include columns (reverse of exclude command)

Syntax:

```
[format] inc[lude]
[format] inc[lude] <colist>
[format] inc[lude] only <colist>
```

With the first form all columns, inclusive all previously excluded columns, are display. With the second form all columns specified in <colist> have been previously excluded and now will be displayed again. With the third form all columns will be excluded and only those specified in <colist> will be displayed. This has the same effect as the 'exclude all but' command.

Default:

all columns are included.

7.3 Output commands

The output commands are intended to print or to save the result in a file using the format as shown on the screen.

7.3.1 Store query result into a file

Syntax:

```
f[ile]
f[ile] <filename>
```

The first form stores the query result into the file UFI.FILE, the second form into the file <filename> in the current working directory. The result is stored as a text file using the current display format as shown on the screen. This is intended for manipulating query results afterwards by any UNIX text processing tool (e.g. copying the result into a document).

After issueing "file" the query result is not shown on the screen. If you want to see the query result on screen, too, an explicit statement 'display' must be issued.

7.3.2 Display Command

Syntax:

```
display
```

If the display command is issued after a SELECT statement, the result tuples are displayed on the. By default, result tuples are always displayed on the screen; only if the result tuples are copied into a file, the display feature has to be activated explicitly.

7.4 Quit command

Syntax:

```
quit
```

The quit command leaves the display menu as does the ESC key. The quit command is more important in procedures when you don't want to see the query result at the terminal but just want to print or store it. In this case you put a sequence of display commands in the procedure immediately behind the "select" statement and terminate the sequence with a print (or store) followed by the quit command. See section 8. for an example.

8 Procedures and Transactions

An UFI procedure may contain multiple queries and data definition statements of TransBase

intermixed with UFI commands, like the format and set statement. Together with the echo and read commands, described later in this section, you get a kind of (elementary) programming language as a tool to compose UFI procedures. The next table lists all valid statements which may be used to build procedures:

Statement	Description
bt	begin transaction
ct	commit transaction
at	abort transaction
echo	echo message on screen
read	read user input
define	define string variable
call	call other procedures
select insert	TB/SQL (see "TB/SQL Manual") .
update	.
delete	.
create	DDL (see "TB/SQL Manual")
drop	.
spool	.
format	all commands from the display menu (see section (7))
set	set UFI directives
 connect	(see section 6.) connect to a database
disconnect	disconnect from a database
switch	change active database
cdq	change current query directory
cdd	change current data directory
timeout	set timeout value

If your procedure consists of more than one statement you have to consider several rules:

First, each statement must begin in a new line. Second, statements must be terminated by a ';' before the next statement follows. Third, all characters in the same line following the semicolon are ignored, i.e. are considered comment.

Example:

```
select count(*) from suppliers; Count all suppliers
```

Fourth, "select" queries and format statements interact in a special way. As soon as "select" statement is encountered during execution of a procedure the display menu is invoked to show the result on the screen.

You may write any display commands (all statements of section 7 including file, quit and display command belong to this class) immediately behind the "select" queries. The display facility reads and interprets all statements until a statement is encountered which is not a display command. Your query result will be displayed in the new format. In addition to the commands started in the procedure, you can enter format commands interactively from the terminal. As soon as you press the ESC key, display is finished and the procedure is resumed.

Often, you want to print preformatted reports which shouldn't be displayed on the screen at all.

In such cases it is more convenient to terminate the sequence of format commands by a quit statement, as illustrated in the next example.

Example:

```
; Print ta nicely formatted report of your relations

          select
t.tname,t.ttype,c.cname,c.cpos,c.ctype,c.indexno
          from systable t,syscolumn c
          where t.tname=c.tname
               and (t.tname<>"systable" and
               t.tname<>"syscolumn" and
               t.tname<>"sysview")
          order by t.tname,c.cpos;

          format exclude cpos;
          format col tname wid 14 name Relation;
          format col ttype wid 3 name R/V;
          format col cname wid 18 name Attribute;
          format col ctype wid 4 name Type;
          format col indexno wid 5 name Index;
          print copies 3;
```

The procedure currently in the editor buffer is started from the main menu by pressing the 'x' key.

If you want to execute a procedure stored in a file, you have to read it in first. As execution proceeds each statement is echoed at the terminal. At any time you can stop execution by pressing the '^\' key (UNIX signal SIGQUIT).

Note:

The UNIX signal SIGINT (DEL key) is ignored during UFI sessions.

Any faulty statement will stop execution, too. In both cases the editor is invoked and the cursor is pointing to the beginning of the statement which has been interrupted. After correcting the error you must leave the editor and restart the procedure once again from the main menu.

8.1 Text variables

The three commands "echo", "read" and "define" provide a basic input/output mechanism in procedures similar to shell procedures.

Syntax:

```
echo <text>
read <variable>
define <variable> <text>
```

The echo statement displays <text> on the screen. The read statement reads a line from the terminal and assigns the input to <variable>. Using the define statement <text> also is assigned to the <variable>.

Variables are used for text expansion. Anywhere in a procedure you may write $<variable>$, in order to substitute for <text> assigned to this <variable>.

Example:

```
Assume we have assigned a string

          define TAS TransAction Software

to the variable TAS. The query

          select * from suppliers s
          where s.name = "$TAS$"

will be expanded to

          select * from suppliers s
          where s.name = "TransAction Software"
```

Expansion may even be nested. In the next example a user is asked to select one of three queries which will be executed.

Example:

```
define a "select * from systable s";
define b "select * from syscolumn s";
define c "select * from sysview s";

echo Select one of the following queries;
echo a - $a$;
echo b - $b$;
echo c - $c$;
echo Enter your choice ?;

read choice;
$$choice$$;
```

The last line is a bit tricky: first, '$choice$' is expande to the letter entered by the user resulting in 'a', 'b' or 'c', respectively. This is further expanded to the corresponding "select" statement.

8.2 Calling other procedures

Within a procedure you may call another procedure. After execution of this procedure control is returned to the calling procedure.

Syntax:

```
call <filename>
```

The procedure in the file <filename> in the current query directory is executed.

The call statement is a way to overcome the limitation of the editor, which doesn't allow for procedures containing more than 300 lines.

Parameters and results can be passed between procedures via common variables. Obviously, relations can be used for data transfer, too.

8.3 Transaction control

A transaction is a group of statements of both the TransBase query language (TB/SQL) or the data definition language (DDL). As far as the database system is concerned, a transaction is handled as an atomic unit. All changes made during a transaction are either written to the database (commit) or the changes are undone and the database is in the same state as before that transaction (abort). The group of statements is enclosed in a 'begin transaction' and 'commit transaction' or 'abort transaction' statement.

Syntax:

```
bt
at
ct
```

The bt command starts a new transaction and exists just for completeness. There is absolutely no need to open explicitly a new transaction, because UFI opens transactions on demand. Further, you must not nest transactions that means you must not issue another bt command while the current transaction is open. It's best never to issue a bt command and leave things for UFI to arrange.

The ct command closes a transaction and all changes are saved in the database. An error will occur if no transaction is open.

The at command also closes a transaction, but all changes are undone and lost! Stopping a transaction with the '^\\' key (signal SIGQUIT) has the same effect as an abort.

The bt, ct and at commands are intended for the user, who wants to have full control over her/his transactions. This user must have set the autocommit directive to off. Additionally, UFI offers automatic control (autocommit=on) of transactions for the casual user. When autocommit is on, UFI takes several criteria into account whether to commit or abort a transaction:

Note:

At the end of a procedure the transaction is also closed and committed.

If an error occurs with a procedure, UFI distinguishes between so called hard and soft errors. The class of soft errors encloses all kind of errors caused by erroneous statements and faulty user input, e.g. a misspelled keyword. The class of hard errors contains all situations (e.g. no space on disk), where TransBase is unable to continue and has to abort the transaction. In case of soft errors the transaction is resumed.

There are several pro and cons concerning explicit vs. automatic control of transactions. First, manual control offers more freedom to include many procedures in one transactions, or to have many transactions in one procedure. Second, you can abort transactions at will, especially as you discover that you have made a severe error, e.g. deleted the wrong relation. On the other hand, unexperienced users often forget to commit their last transaction. While one is warned before leaving UFI, another situation might occur where a hard error automatically aborts the whole transaction, so that all work was done in

vain.

8.4 Database Control

Since TransBase allows remote database access and distributed transactions, UFI provides for multiple databases, too. To not confuse the user, UFI defines a "current database" to which all statements refer. Statements are provided to connect to a database, to disconnect from a database and to change the current database.

SYNTAX:

```
connect dbname [ login ] [ password ]

disconnect dbname

switch dbname
```

Databases are always identified by their logical names. A logical database name is either a local database name or a local database name followed by an ampersand character (@) followed by a host name.

If the login and/or the password are omitted in the connect statement, the user will be prompted for them.

The disconnect statement disconnects from the current database. If a transaction is open (with autocommit switched off) the user will be prompted.

The switch statement changes the current database to another database.

8.5 Changing Current Directories

UFI distinguishes three "current directories", namely the:

current working directory
current query directory
current data directory.

SYNTAX:

```
cdq pathname

cdd pathname
```

The current working directory is not changed within the UFI session; the current working directory is inherited from the calling SHELL.

The current query directory is used to store and recall procedures.

The current data directory is used to locate external files which are spooled into the database or are spooled from the database.

The pathname to be specified with the cdd and cdq commands is either an absolute pathname or is a relative pathname; in the latter case it refers to the current data directory or to the current query directory, resp.

8.6 Setting Timeout

If a statement cannot be executed due to locks held by other transactions, a timeout mechanism is used to prevent from indefinite delay.

SYNTAX:

```
timeout seconds
```

The timeout period is set to the number given as parameter. This setting is used for all databases.

By default, the timeout period is set to 60 seconds.

9 Error handling and trouble shooting

Both, UFI and TransBase detect various errors which are immediately displayed at the user's terminal. You must press ESC to continue the session after an error message is displayed.

UFI errors are displayed in the form

```
UFI reports an error 9: You made an error
```

Whenever TransBase detects an error, you will see an error message of the following form at your terminal:

```
TransBase reports error (9999): /* something has gone wrong */
```

As mentioned at the beginning of section 2, TransBase writes diagnostic messages into a file. These messages are supplementary to the error messages as seen by the user. They describe the runtime state of TransBase at the moment the error happened, which is primarly intended for analysing severe system problems.

You will never have to look for the diagnostic messages, except when you believe that something has gone wrong in an unforeseen way. This diagnostic file is intended to support TransAction Software in finding what has gone wrong.